Running OpenFOAM in parallel

Tommaso Lucchini

Department of Energy
Politecnico di Milano

Running in parallel

- The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution.
- The first step is to decompose the domain using the `decomposePar` utility.
- The user must edit the dictionary associated with decomposePar named `decomposeParDict` which is located in the system directory of the case.
- The first entry is `numberOfSubdomains` which specifies the number of sub-domains into which the case will be decomposed, usually corresponding to the number of processors available for the case.
- The parallel running uses the public domain openMPI implementation of the standard message passing interface (MPI).
- The process of parallel computation involves: decomposition of mesh and fields; running the application in parallel; and, post-processing the decomposed case.

Running in parallel

- We shall use the pitzDaily case for the simpleFoam application tutorial:

```
tut
cd simpleFoam
cd pitzDaily
```

- We can copy the pitzDaily case to the pitzDailyParallel:

```
cp -r pitzDaily $FOAM_RUN/pitzDailyParallel
```

- And copy into the system directory the `decomposeParDict` file available in the pitzdailtExptInlet case:

```
cp pitzDailyExptInlet/system/decomposeParDict pitzDailyParallel/sys
```

Running in parallel: domain decomposition

- The mesh and fields are decomposed using the `decomposePar` utility.
- The underlying aim is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution.
- The geometry and fields are broken up according to a set of parameters specified in a dictionary named `decomposeParDict` that must be located in the `system` directory of the case of interest.
- In the `decomposeParDict` file the user must set the number of domains which the case should be decomposed into: usually it corresponds to the number of cores available for the calculation:

  ```
  numberOfSubdomains 2;
  ```

Running in parallel: domain decomposition

- The user has a choice of four methods of decomposition, specified by the method keyword.
- For each method there are a set of coefficients specified in a sub-dictionary of decompositionDict, named `<method>Coeffs`, used to instruct the decomposition process:

  ```
  method simple/hierarchical/metis/manual;
  ```

  - **simple**: simple geometric decomposition in which the domain is split into pieces by direction, e.g. 2 pieces in the x direction, 1 in y etc.
  - **hierarchical**: Hierarchical geometric decomposition which is the same as simple except the user specifies the order in which the directional split is done, e.g. first in the y-direction, then the x-direction etc.
  - **metis**: METIS decomposition which requires no geometric input from the user and attempts to minimize the number of processor boundaries. The user can specify a weighting for the decomposition between processors which can be useful on machines with differing performance between processors.
  - **manual**: Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor.

Running in parallel: domain decomposition

- The coefficients sub-dictionary for the `simple` method are set as follows:

```
simpleCoeffs
{
    n                   (2 1 1);
    delta               0.001;
}
```

  Where **n** is the number of sub-domains in the x, y, z directions (nx ny nz) and **delta** is the cell skew factor, typically 10^{-3}.

- The coefficients sub-dictionary for the `hierarchical` method are set as follows:

```
hierarchicalCoeffs
{
    n                   (2 2 1);
    delta               0.001;
    order               xyz;
}
```

  Where **n** is the number of sub-domains in the x, y, z directions, **delta** is the cell skew factor and **order** is the order of decomposition xyz/xzy/yxz...

Running in parallel: domain decomposition

- The coefficients sub-dictionary for the `metis` method are set as follows:

```
metisCoeffs
{
    processorWeights
    (
        1
       ...
        1
    );
}
```

  It is the list of weighting factors for allocation of cells to processors: `<wt1>` is the weighting factor for processor 1, etc.

- The coefficients sub-dictionary for the `manual` method contains the reference to a file:

```
manualCoeffs
{
    dataFile        "<fileName>";
}
```

  It is the name of file containing data of allocation of cells to processors.

Running in parallel: domain decomposition

- Data files may need to be distributed if only local disks are used in order to improve performance.
- The root path to the case directory may differ between machines. The paths must then be specified in the decomposeParDict dictionary using `distributed` and `roots` keywords.
- The distributed entry should read:

  `distributed yes;`

  and the roots entry is a list of root paths, `<root0>`, `<root1>`, . . . , for each node:

```
roots
<nRoots>
(
"<root0>"
"<root1>"
...
);
```

  where `<nRoots>` is the number of roots.
- Each of the processorN directories should be placed in the case directory at each of the root paths specified in the decomposeParDict dictionary.

Running in parallel: domain decomposition

- The decomposePar utility is executed in the normal manner by typing:

  `decomposePar -case [caseName]`

- The output will look like:

```
Calculating distribution of cells
Selecting decompositionMethod metis
...
Processor 0
    Number of cells = 6113
    Number of faces shared with processor 1 = 58
    Number of processor patches = 1
    Number of processor faces = 58
    Number of boundary faces = 12514

Processor 1
    Number of cells = 6112
    Number of faces shared with processor 0 = 58
    Number of processor patches = 1
    Number of processor faces = 58
    Number of boundary faces = 12496
```

Running in parallel: domain decomposition

- On completion, a set of subdirectories have been created, one for each processor, in the case directory.
- The directories are named processorN where N = 0, 1, ... represents a processor number and contains a time directory, containing the decomposed field descriptions, and a constant/polyMesh directory containing the decomposed mesh description.

```
pitzDailyParallel
      |------> 0
      |------>  constant
      |------>  processor0
      |------>  processor1
      |------>  system
```

- *openMPI* can be run on a local multiprocessor machine very simply but when running on machines across a network, a file must be created that contains the host names of the machines. The file can be given any name and located at any path.

Running in parallel: execution

- An application is run in parallel using the `mpirun` command:

  ```
  mpirun --hostfile <machines> -np <nProcs> <foamExec>
  <root> <case> <otherArgs> -parallel > log
  ```

  where: `<nProcs>` is the number of processors; `<foamExec>` is the executable, e.g. simpleFoam; and, the output is redirected to a file named log.

- The `<machines>` file contains the names of the machines listed, one machine per line.

- The names must correspond to a fully resolved hostname in the /etc/hosts file of the machine on which the openMPI is run. The `<machines>` file would contain:

  ```
  hostname1
  hostname2 cpu=2
  hostname3
  ```

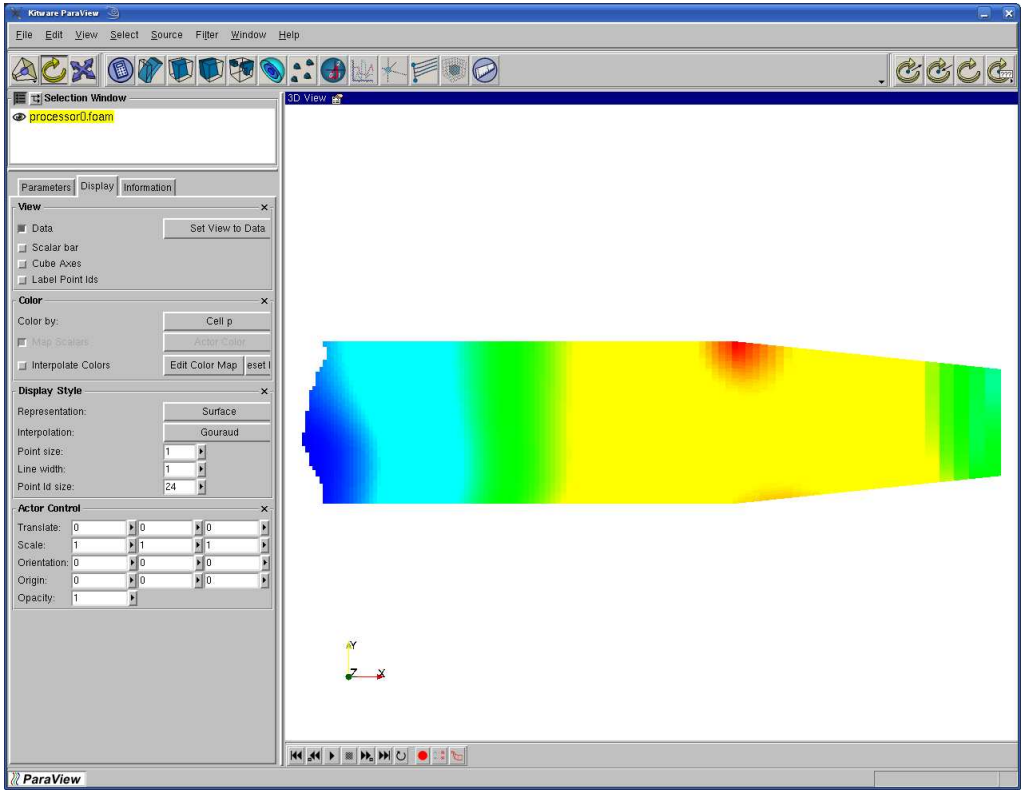
- In our case it becomes:

  ```
  mpirun -np 2 simpleFoam -case $FOAM_RUN/pitzDailyParallell \
  -parallel > log
  ```

Running in parallel: post-processing

When post-processing cases that have been run in parallel the user has two options:

- reconstruction of the mesh and field data to recreate the complete domain and fields, which can be post-processed as normal;
- post-processing each segment of decomposed domain individually;



Running in parallel: post-processing

- After a case has been run in parallel, it can be reconstructed for post-processing by merging the sets of time directories from each processorN directory into a single set of time directories.
- The reconstructPar utility performs such a reconstruction by executing the command:

```
reconstructPar -case [case]
```

- In our case it will become:

```
reconstructPar -case pitzDailyParallel
Time = 0
Reconstructing FV fields
    Reconstructing volScalarFields
        p
      .....
    Reconstructing volVectorFields
        U
    Reconstructing volTensorFields
        R
No point fields
No lagrangian fields
```